Rendu TD4

BILLEQUIN Thomas, DAVAL Enzo, HATTIGER Louis

Question 1:

Tout est opérationnel.

Question 2:

Lorsqu'un obstacle est détecté à une certaine distance -> khepera III s'arrête (vitesse du moteur gauche et droit à 0)
Si non -> khepera III avance tout droit (vitesse du moteur gauche et droit à max_speed)
Voilà ce que ça donne en code:

```
for (int i = 0; i < num_sensors; i++) {
  sensor_values[i] = wb_distance_sensor_get_value(sensors[i]);
  //printf("valeur sensor %d: %f \n", i, sensor_values[i]);
}
int obstacle_detected = 0;
for (int i = 0; i < num_sensors; i++) {
  if (sensor_values[i] > 300) {
    obstacle_detected = 1;
    break;
  }
}
if (obstacle_detected) {
  wb_motor_set_velocity(left_motor, 0.0);
  wb_motor_set_velocity(right_motor, 0.0);
} else {
  wb_motor_set_velocity(left_motor, max_speed);
  wb_motor_set_velocity(right_motor, max_speed);
}
```

+ video

code alphabot:
On s'appuie sur l'abonnement aux topics des capteurs de proximité. Il faut au préalable lancer le micro-service mqtt avec docker-compose up -d. Le programme publie des instructions aux actionneurs de mqtt, dans ce cas aux moteurs.

```python
from ab2_mqtt import MQTTClient

def move_forward(client, userdata, msg):
        global left_obstacle, right_obstacle
        if msg.topic == client.TOPIC_OBSTACLE_LEFT:
                left_obstacle = msg.payload.decode('utf8') == 'True'
                print("Obstacle left: ", msg.payload.decode('utf8'))
        elif msg.topic == client.TOPIC_OBSTACLE_RIGHT:
                right_obstacle = msg.payload.decode('utf8') == 'True'
                print("Obstacle right: ", msg.payload.decode('utf8'))

        if left_obstacle or right_obstacle:
                client.publish(client.TOPIC_MOTORS, "0 0")

        elif not left_obstacle and not right_obstacle:
                client.publish(client.TOPIC_MOTORS, "25 25")

# Parse environment variables to get MQTT broker parameters
DOCKER_VARENV = ['MQTT_HOST', 'MQTT_PORT']

if set(DOCKER_VARENV).issubset(set(os.environ)):
    MQTT_HOST = env(DOCKER_VARENV[0], default='localhost')
    MQTT_PORT = env.int(DOCKER_VARENV[1], default=1883)
else:
    MQTT_HOST = 'localhost'
    MQTT_PORT = 1883

# Start a connexion to the MQTT broker
QOS = 0
client = MQTTClient("controler", MQTT_HOST, MQTT_PORT, QOS)
client.loop_start() # start client in a dedicated thread to

left_obstacle = False
right_obstacle = False

client.publish(client.TOPIC_MOTORS, "25 25")
# Subscribe to a topic to get IR sensors values
client.message_callback_add(client.TOPIC_OBSTACLE_LEFT, move_forward)
client.message_callback_add(client.TOPIC_OBSTACLE_RIGHT, move_forward)

while True:
        time.sleep(1)

# End connexion to MQTT broker before exiting
client.loop_stop()
```

Vidéo:
Concernant cette vidéo et la suivante, je n'avais pas chez moi les bonnes conditions pour tester l'alphabot hors de son socle (cable éthernet d'un côté, alimentation de l'autre). L'obstacle détecté est donc ma main.

Question 3:

J'ai repris le code qui était fourni à la base pour éviter les obstacles.
Dans ce code on remarque que la vitesse du moteur gauche et la vitesse du moteur droit sont instanciés au début à 0. Puis on associe une valeur à ces vitesses avec cette formule :

```
speed[i] += speed_unit * matrix[j][i] * (1.0 - (sensors_value[j] / range));
```

Si on prend la formule donnée dans le td sur la partie évitement d'obstacles

$$V = k * \sum_{i=0}^{n} Wi * Xi$$

On associe k à speed_unit, le poids Wi à la matrice et les valeurs renvoyées par les capteurs à `(1.0 - (sensors_value[j] / range))`
Speed[i] avec i allant de 0 à 1 correspond respectivement à la vitesse du moteur gauche et du moteur droit.

Matrix[j][i] ci-dessous représente les différents poids affectés au capteurs

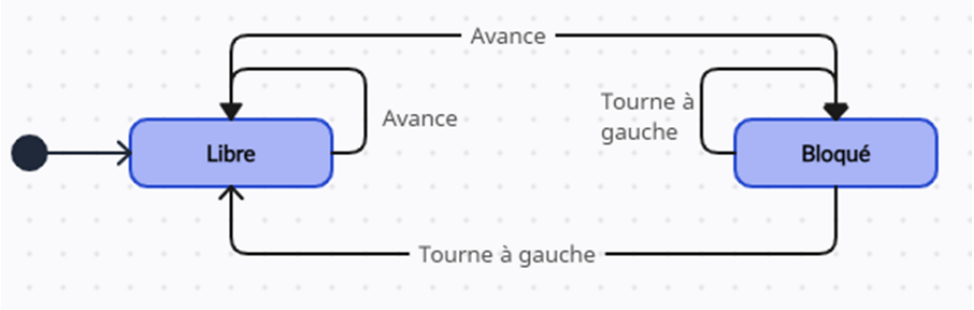
```
{{-5000, -5000},  {-20000, 40000}, {-30000, 50000}, {-70000, 70000}, {70000, -60000}
 {50000, -40000}, {40000, -20000}, {-5000, -5000},  {-10000, -10000}};
```

Question 4 :

Je commencerai par mettre en forme les données afin de bien recevoir des données binaires des capteurs puis je transmettrai ces données au réseau de neurones afin qu'il parcoure la plus grande distance possible.
L'alphabot possède 2 capteurs qui sont des détecteurs d'obstacles ainsi qu'un moteur gauche et un moteur droit. On aurait donc pour les poids une matrice carré d'ordre 2, M.
L'équation associé à l'alphabot pour les vitesses :
V1 correspond à la vitesse du moteur droit et V2 correspond à la vitesse du moteur gauche.

$$V1 = k * State_l * \sum_{i=1}^{2} M_{1i} \qquad V2 = k * State_r * \sum_{i=1}^{2} M_{2i}$$

Question 5:



Le code de cet automate est observable dans automate.c.
Le code de l'alphabot reste identique à celui de la question 2, sauf que cette fois au lieu de s'arrêter lorsqu'un obstacle est détecté on change la puissance des moteurs pour qu'il puisse tourner à gauche (on met donc plus de puissance dans le moteur de droite).

```
def move_forward(client, userdata, msg):
        global left_obstacle, right_obstacle
        if msg.topic == client.TOPIC_OBSTACLE_LEFT:
                left_obstacle = msg.payload.decode('utf8') == 'True'
                print("Obstacle left: ", msg.payload.decode('utf8'))
        elif msg.topic == client.TOPIC_OBSTACLE_RIGHT:
                right_obstacle = msg.payload.decode('utf8') == 'True'
                print("Obstacle right: ", msg.payload.decode('utf8'))

        if left_obstacle or right_obstacle:
                client.publish(client.TOPIC_MOTORS, "10 50")
        elif not left_obstacle and not right_obstacle:
                client.publish(client.TOPIC_MOTORS, "25 25")
```

Vidéo:
On peut voir que lorsqu'un obstacle est détecté la vitesse du moteur gauche diminue et celui du moteur droit augmente pour pouvoir tourner à gauche. Lorsqu'il n'y a plus d'obstacles détectés l'alphabot avance tout droit.

Question 6:

Il faudrait modifier les poids des capteurs pour réduire le poids à droite, faisant donc tourner le robot à gauche et donc longer l'obstacle à droite :

- Faire toujours tourner le robot a gauche :
  -> Tourner à gauche signifie vitesse de droite > vitesse de gauche
  -> speed[0] < speed[1]
  -> matrix[j][0] < matrix[j][1]

- Ne pas le faire + tourner que nécessaire :
  -> Regarder les capteurs de devant (2 et 3) et tourner tant qu'il sont > 500

Code :

```
226         if (sensors_value[5] > 500 || sensors_value[4] > 500){ //obstacle proche
227           printf("Faire tourner a gauche! %d ; %f ; %f\n",counter,sensors_value[3],sensors_value[4]);
228           if (matrix[j][0] > matrix[j][1]) {
229             if (i == 0){
230               speed[0] += speed_unit * matrix[j][1] * (1.0 - (sensors_value[j] / range));
231             } else {
232               speed[1] += speed_unit * matrix[j][0] * (1.0 - (sensors_value[j] / range));
233             }
234
235           }else{
236              speed[i] += speed_unit * matrix[j][i] * (1.0 - (sensors_value[j] / range));
237           }
238         } else {
239         /*
240         On veut que lorsqu'un obstacle est detecté, le robot tourne a gauche et longe l'obstacle,
241         pour ce faire :
242         - Faire toujours tourner le robot a gauche :
243           -> Tourner a gauche signifie vitesse de droite > vitesse de gauche
244           -> speed[0] < speed[1]
245           -> matrix[j][0] < matrix[j][1]
246
247         - Ne pas le faire + tourner que necessaire :
248           -> Regarder les capteurs de devant (2 et 3) et tourner tant qu'il sont > 500
249         */
250         speed[i] += speed_unit * matrix[j][i] * (1.0 - (sensors_value[j] / range));
251         }
252       }
```

Le code est également observable dans droite.c

Video : Q6.mp4

Question 7:

A partir d'un capteur et d'un émetteur infrarouge, le robot peut identifier s' il y a une ligne en dessous par les valeurs récupérées sous le format d'un array. On va prendre ici un array de 5 capteurs placé sous le robot avec les poids suivants.

-2	-1	0	1	2

Ainsi:

$$\begin{pmatrix} V_r \\ V_l \end{pmatrix} = V_b \cdot \begin{pmatrix} 2 & 1 & 0 & -1 & -2 \\ -2 & -1 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{pmatrix}$$

Avec Vr et Vl les vitesses respectives des roues, Vb la vitesse de base, la matrice centrale avec toutes les valeurs correspond aux poids et Xn aux valeurs des capteurs (des flottants allant de 0 à 1000).

Code alphabot:

```python
def follow_line(client, userdata, msg):
        ir_data = msg.payload.decode('utf8').split(' ')

        position = float(ir_data[0])  # Récupérer la position de la ligne noire
        sensor_values = [float(value.strip('[],')) for value in ir_data[1:]]  # Récupérer les valeurs des capteurs

        print("Position: ", position)
        print("Sensors values: ", sensor_values)
        # Définir les matrices pour ajuster la vitesse des moteurs en fonction des valeurs des capteurs
        motor_weights = np.array([[2, 1, 0, -1, -2], [-2, -1, 0, 1, 2]])
        sensor_values_matrix = np.array(sensor_values).reshape((-1, 1))  # Transformer les valeurs des capteurs en matrice colonne

        # Calculer les vitesses des moteurs en fonction des valeurs des capteurs et de la position de la ligne noire
        motor_speeds = np.dot(motor_weights, sensor_values_matrix) * 25  # Adapter 25 selon la vitesse de base

        # Appliquer les vitesses des moteurs en fonction des valeurs des capteurs
        left_motor_speed = int(motor_speeds[0][0]//100)
        right_motor_speed = int(motor_speeds[1][0]//100)
        print("Vitesse  moteur gauche: ", left_motor_speed)
        print("Vitesse moteur droit: ", right_motor_speed)
        # Publier les vitesses des moteurs
        client.publish(client.TOPIC_MOTORS, f"{left_motor_speed} {right_motor_speed}")
        time.sleep(0.5)
```

Pas de vidéo pour cette partie, pas de possibilités de créer un circuit avec une ligne noire sur une route blanche.

Question 8:

Si l'on reprend la question 6 et le code que nous avons fourni, une stratégie de coordination est déjà mise en place puisque le robot utilise l'algorithme de braintenberg pour se déplacer et dans certaines conditions, c'est notre algorithme qui prend le dessus pour faire du contour d'obstacle par la droite. En modifiant un petit peu le code, on retrouve rapidement une stratégie de coordination par vote.

Code :

```
213       int vote_braintenberg = 500;
214       int vote_contour = 500;
215       //if (sensors_value[j] > 500) printf("Capteur %d -> distance proche\n",j);
216         /*
217          * We need to recenter the value of the sensor to be able to get
218          * negative values too. This will allow the wheels to go
219          * backward too.
220          */
221         /*if (sensors_value[j] > 500.0){
222           speedV1 = 0;
223           blockingIdex = j;
224          }*/
225        /*double left_weight = matrix[j][0];
226        double right_weight = matrix[j][1];*/
227        if (sensors_value[5] > 500) vote_contour += 125;
228        else vote_contour -= 100;
229
230        if (sensors_value[4] > 500) vote_contour += 125;
231        else vote_contour -= 100;
232        //Capteur 4
233
234        if (vote_contour > vote_braintenberg){ //obstacle proche
235          printf("Faire tourner a gauche! %d ; %f ; %f\n",counter,sensors_value[3],sensors_value[4]);
236          if (matrix[j][0] > matrix[j][1]) {
```

Le code est également observable dans voting.c.
Video : Q8_1.mp4

Le deuxième type de coordination implémenté est le coordination séquentielle, c'est à dire que chaque comportement s'effectue les uns après les autres en reprenant les valeurs du comportement précédent.

Code :

```
226         speed[i] += speed_unit * matrix[j][i] * (1.0 - (sensors_value[j] / range));
227
228
229         if (sensors_value[5] > 500 || sensors_value[4] > 500){ //obstacle proche
230           printf("Faire tourner a gauche! %d ; %f ; %f\n",counter,sensors_value[3],sensors_value[4]);
231           if (matrix[j][0] > matrix[j][1]) {
232             if (i == 0){
233               speed[0] -= 2 * speed[0];//+= speed_unit * matrix[j][1] * (1.0 - (sensors_value[j] / range));
234             } else {
235               speed[1] += 2 * speed[1];//speed_unit * matrix[j][0] * (1.0 - (sensors_value[j] / range));
236             }
237           }
238         }
```

Le code est observable également dans sequentiel.c
Video : Q8_2.mp4